DA-STATS

Introduction

Shan E Ahmed Raza
Department of Computer Science
University of Warwick
shan.raza@warwick.ac.uk

DASTATS – Teaching Team

Shan E Ahmed Raza

Fayyaz Minhas

Richard Kirk

About Me

warwick.ac.uk/searaza

- PhD (Computer Science)
  - ➞ University of Warwick 2011-2014
  - ➞ Supervisor(s): Professor Nasir Rajpoot and Dr John Clarkson
- Research Fellow
  - ➞ University of Warwick 2014 – 2017
- Postdoctoral Fellow
  - ➞ The Institute of Cancer Research 2017 – 2019
- Assistant Professor
  - ➞ Computer Science, University of Warwick 2019 -
- Research Interests
  - ➞ Computational Pathology
  - ➞ Multi-Channel and Multi-Model Image Analysis
  - ➞ Deep Learning and Pattern Recognition

Course Outline

warwick.ac.uk/searaza

- Introduction To Python.
- Introduction To Descriptive And Predictive Techniques.
- Data Visualisation And Reporting Techniques.
- Probability And Bayes Theorem
- Sampling From Univariate Distributions.
- Concepts Of Multivariate Analysis.
- Linear Regression.
- Application Of Data Analysis Requirements In Work.
- Appreciate Of Limitations Of Traditional Analysis.

DA-STATS

Topic 01: Introduction to Python

Shan E Ahmed Raza

Department of Computer Science

University of Warwick

shan.raza@warwick.ac.uk

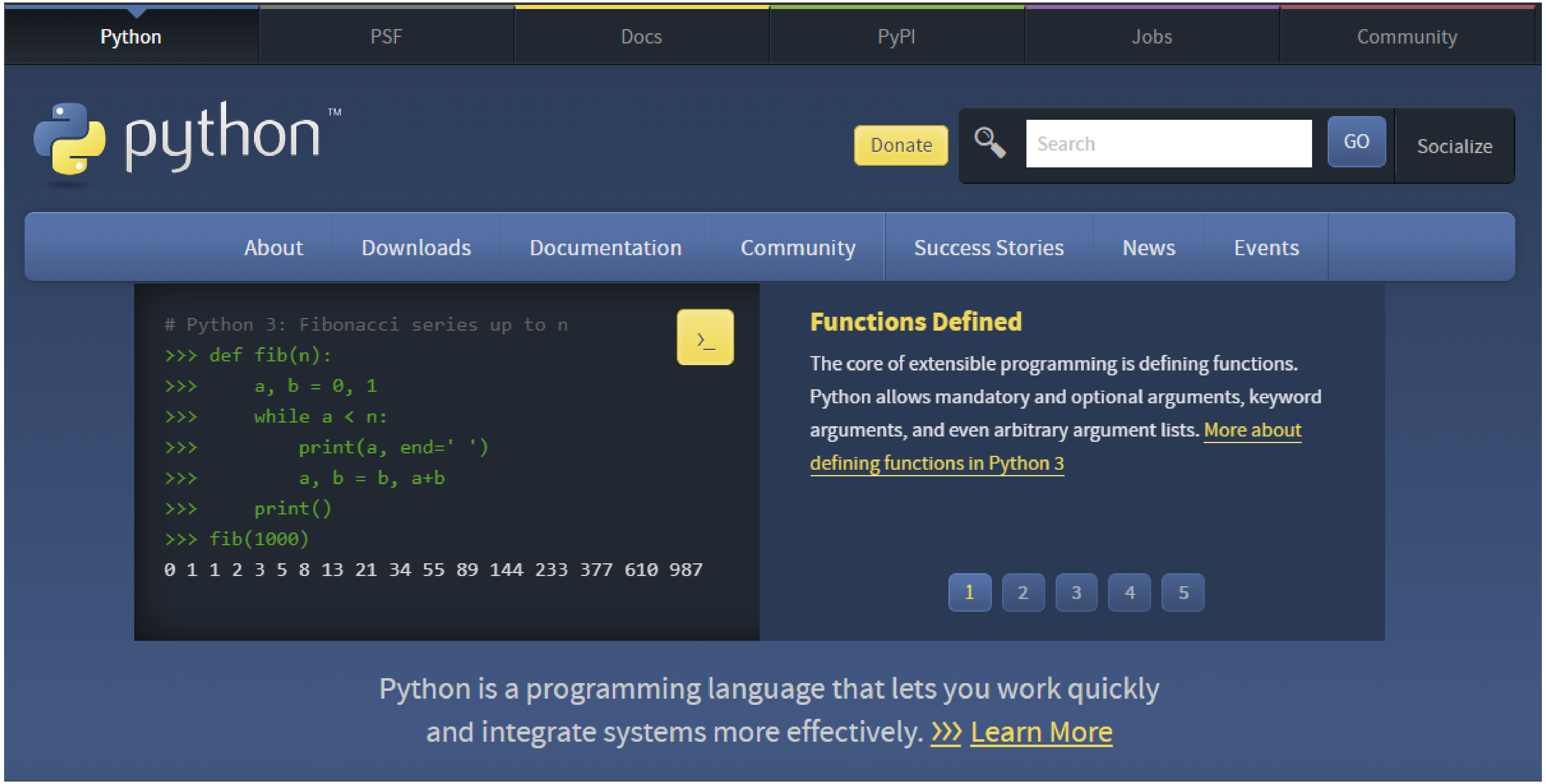
Introduction to Python

- Python is an interpreted, interactive and object-oriented language
- C/C++/Java libraries but find the usual write, compile, test, re-compile cycle is too slow
- *Extensible*
- very-high-level language
- statement grouping done by indentation
- Supports all Operating Systems
  - ➞Windows, Mac and Unix

https://docs.python.org/3/download.html

Python - installation

WARWICK



```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

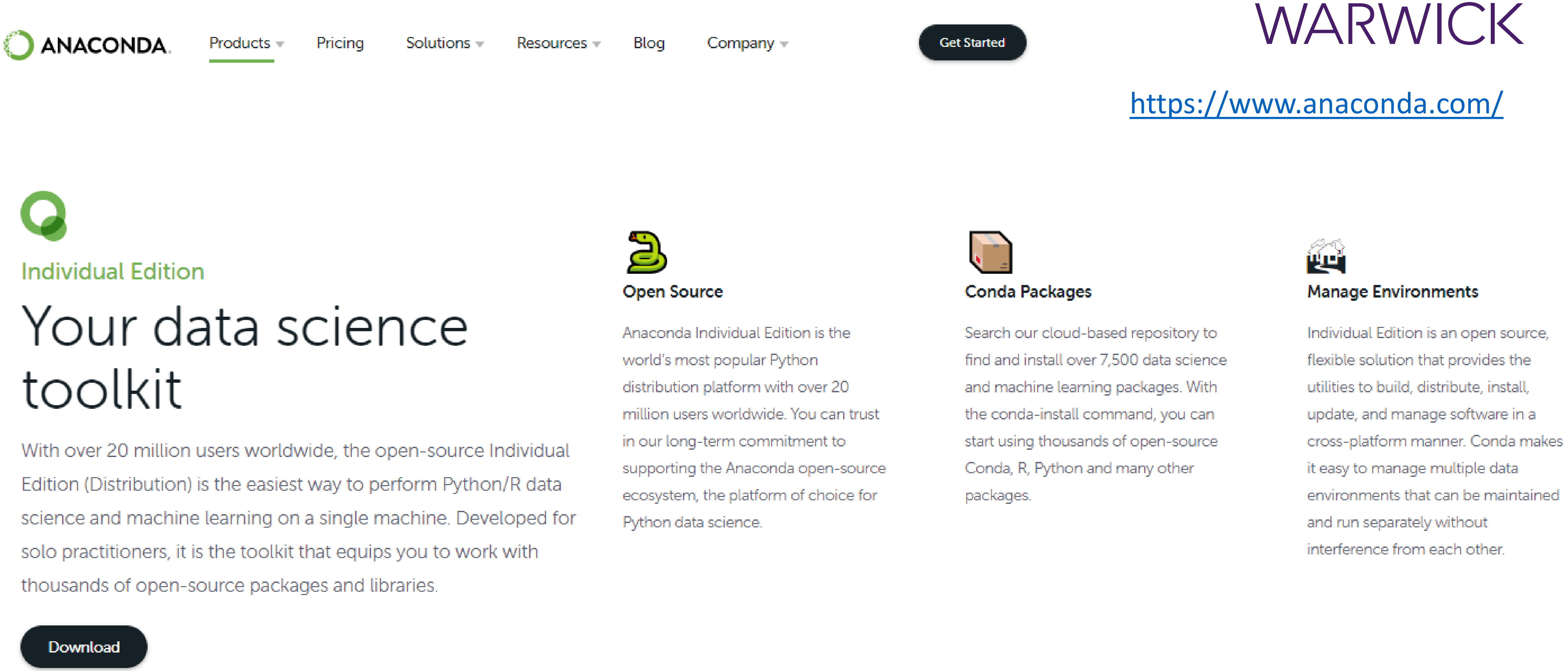
Functions Defined

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. More about defining functions in Python 3

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

https://www.python.org/

Anaconda - installation



https://www.anaconda.com/

Individual Edition

Your data science toolkit

With over 20 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Open Source

Anaconda Individual Edition is the world's most popular Python distribution platform with over 20 million users worldwide. You can trust in our long-term commitment to supporting the Anaconda open-source ecosystem, the platform of choice for Python data science.

Conda Packages

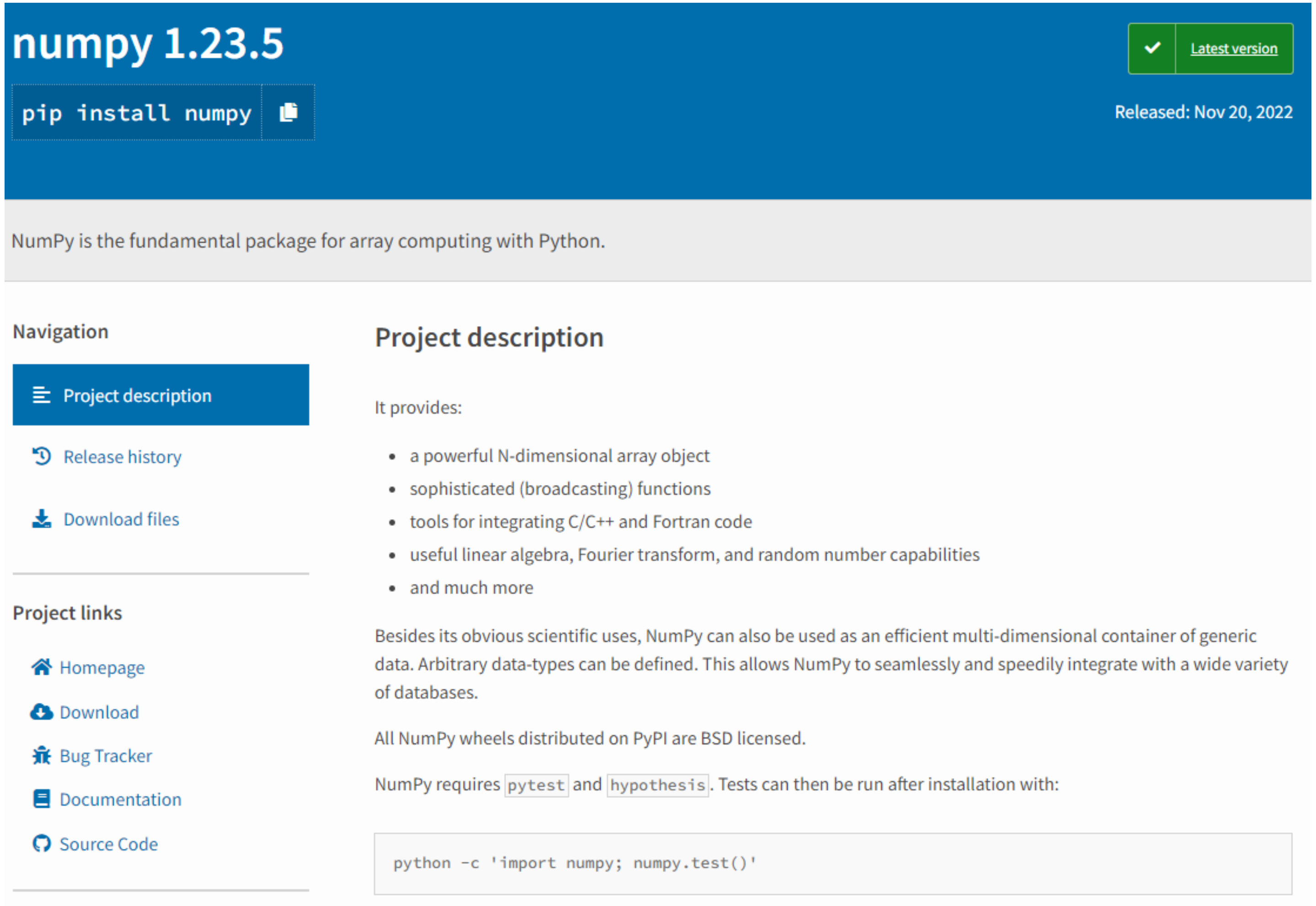
Search our cloud-based repository to find and install over 7,500 data science and machine learning packages. With the conda-install command, you can start using thousands of open-source Conda, R, Python and many other packages.

Manage Environments

Individual Edition is an open source, flexible solution that provides the utilities to build, distribute, install, update, and manage software in a cross-platform manner. Conda makes it easy to manage multiple data environments that can be maintained and run separately without interference from each other.

pypi – pip install

WARWICK



numpy 1.23.5

```
pip install numpy
```

Released: Nov 20, 2022

NumPy is the fundamental package for array computing with Python.

Project description

It provides:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities
- and much more

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

All NumPy wheels distributed on PyPI are BSD licensed.

NumPy requires `pytest` and `hypothesis`. Tests can then be run after installation with:

```
python -c 'import numpy; numpy.test()'
```

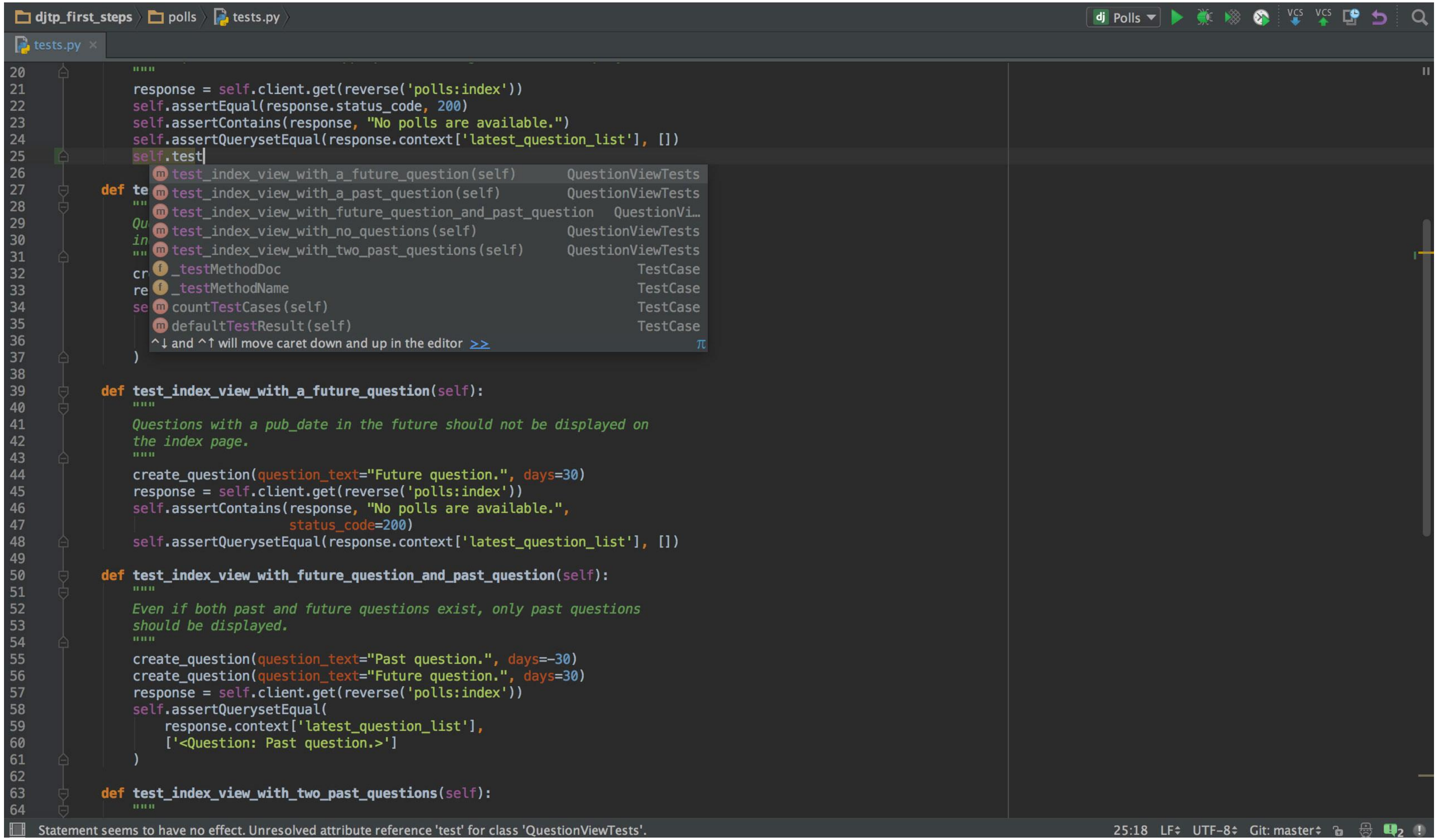
https://pypi.org/project/numpy/

What is NumPy?

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the *ndarray* object. This encapsulates *n*-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

Editors/IDE



```
        """
        response = self.client.get(reverse('polls:index'))
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, "No polls are available.")
        self.assertQuerysetEqual(response.context['latest_question_list'], [])
        self.test

    def te
        """
        Qu
        in
        """
        cr
        re
        se

        )

    def test_index_view_with_a_future_question(self):
        """
        Questions with a pub_date in the future should not be displayed on
        the index page.
        """
        create_question(question_text="Future question.", days=30)
        response = self.client.get(reverse('polls:index'))
        self.assertContains(response, "No polls are available.",
                            status_code=200)
        self.assertQuerysetEqual(response.context['latest_question_list'], [])

    def test_index_view_with_future_question_and_past_question(self):
        """
        Even if both past and future questions exist, only past questions
        should be displayed.
        """
        create_question(question_text="Past question.", days=-30)
        create_question(question_text="Future question.", days=30)
        response = self.client.get(reverse('polls:index'))
        self.assertQuerysetEqual(
            response.context['latest_question_list'],
            ['<Question: Past question.>']
        )

    def test_index_view_with_two_past_questions(self):
        """
```

```
m test_index_view_with_a_future_question(self)        QuestionViewTests
m test_index_view_with_a_past_question(self)          QuestionViewTests
m test_index_view_with_future_question_and_past_question  QuestionVi…
m test_index_view_with_no_questions(self)             QuestionViewTests
m test_index_view_with_two_past_questions(self)       QuestionViewTests
f _testMethodDoc                                               TestCase
f _testMethodName                                              TestCase
m countTestCases(self)                                         TestCase
m defaultTestResult(self)                                      TestCase
^↓ and ^↑ will move caret down and up in the editor >>                π
```

Statement seems to have no effect. Unresolved attribute reference 'test' for class 'QuestionViewTests'.   25:18 LF÷ UTF-8÷ Git: master÷

PyCharm

Visual Studio

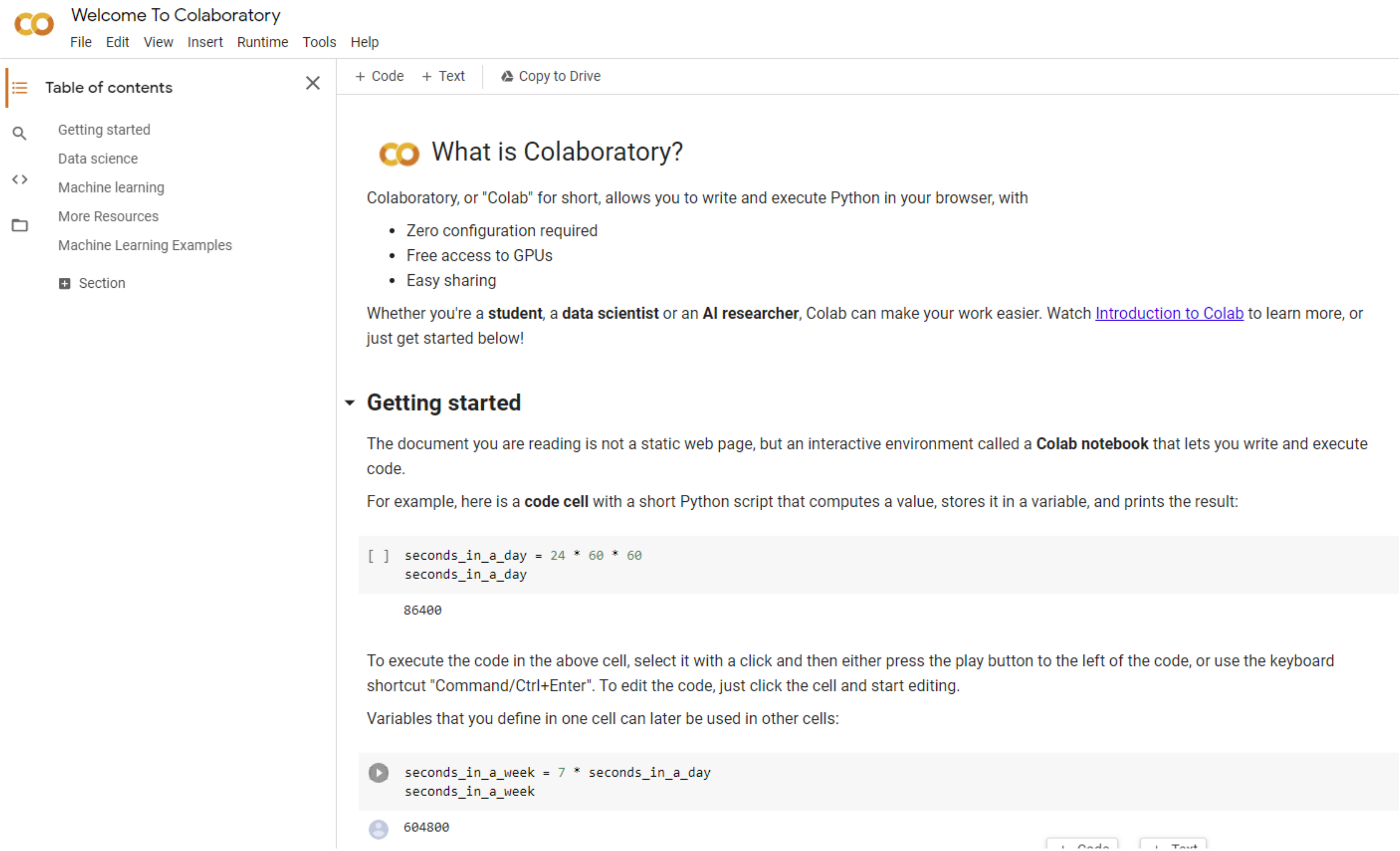
Spyder

.

.

.

Jupyter Notebooks



What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ]  seconds_in_a_day = 24 * 60 * 60
     seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

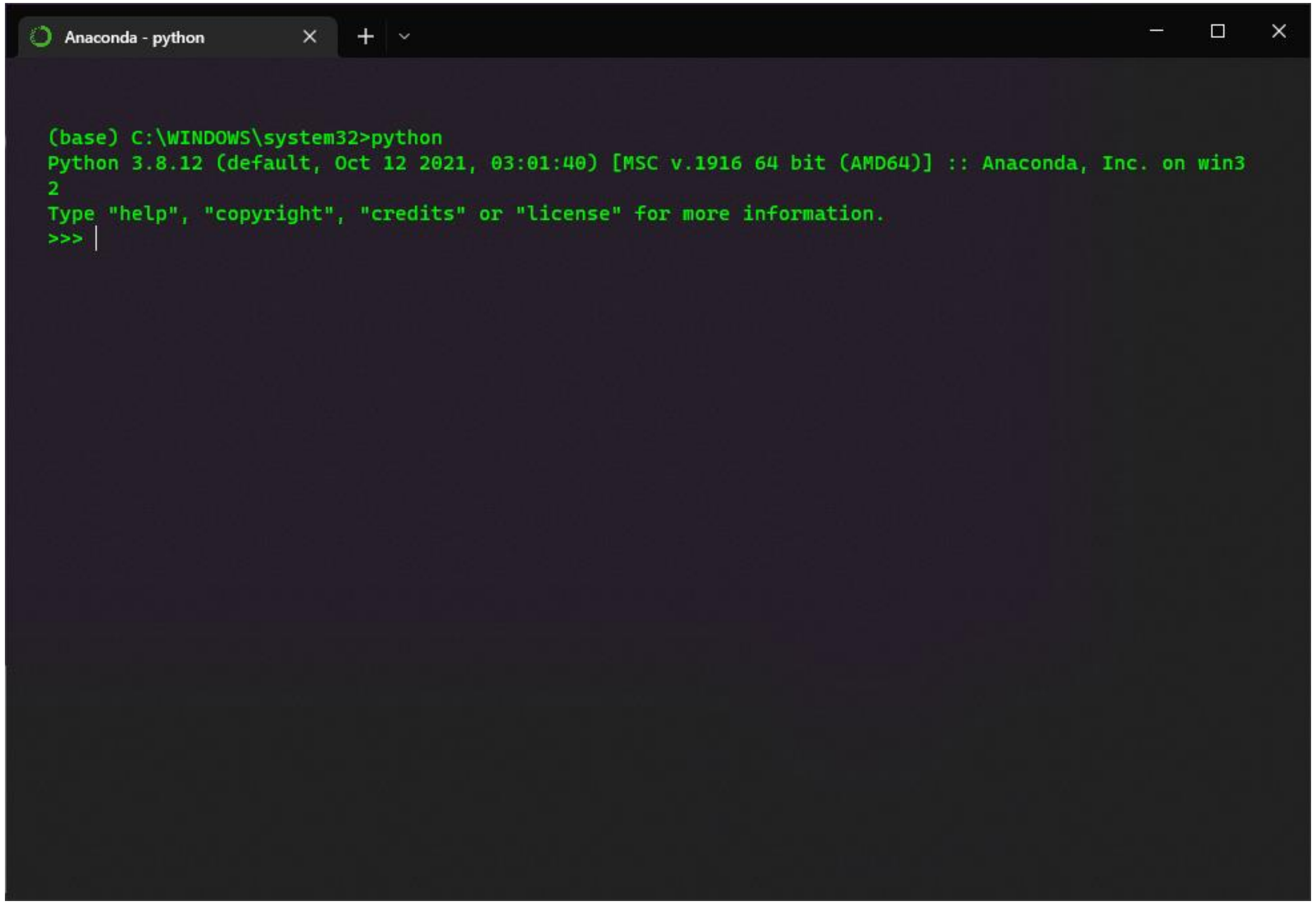
Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Python Version

Type python in terminal or command prompt



```
(base) C:\WINDOWS\system32>python
Python 3.8.12 (default, Oct 12 2021, 03:01:40) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win3
2
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

Basic Syntax

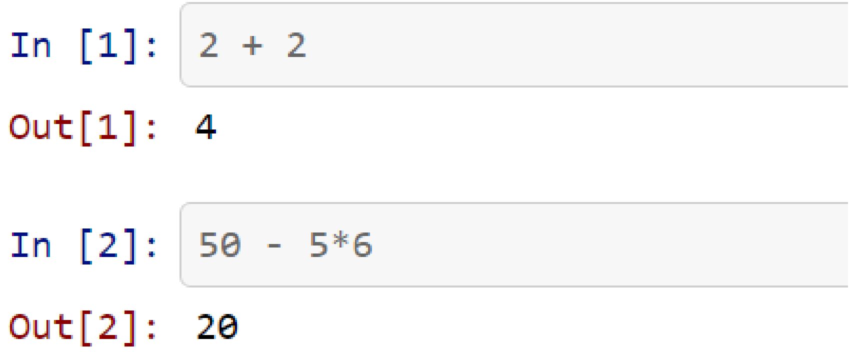
- Python looks like many other modern programming languages such as C, Java, PHP and Perl.
- There are no semi-colons at the end of lines.
- the operators +, -, * and / work just like in most other languages
- parentheses “(())” & Indentation can be used for grouping
- Comments in Python start with the hash character, #, and extend to the end of the physical line.

https://github.com/shaneahmed/StatswithPython/blob/main/Introduction%20to%20Python.ipynb

Using Python as a Calculator

Python interpreter acts as a simple calculator

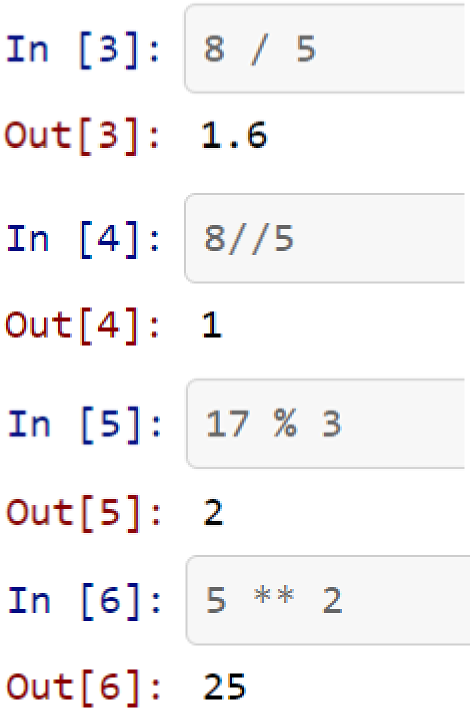
you can type an expression at it and it will write the value.



```
In [1]: 2 + 2

Out[1]: 4

In [2]: 50 - 5*6

Out[2]: 20
```

Using Python as a Calculator

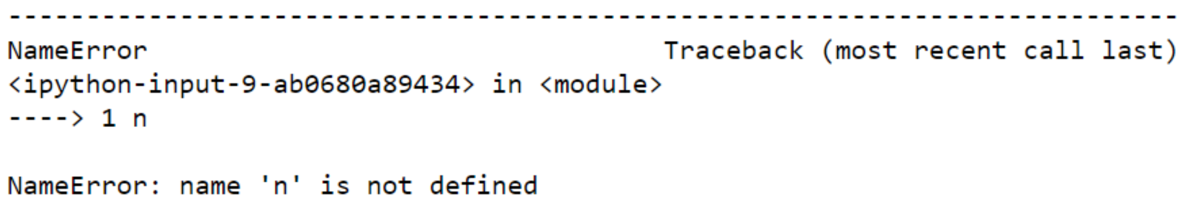
- Division always returns a floating-point number
- Floor division using the operator “//” discards the fractional part
- The “%” operator returns the remainder of the division
- For power calculations use the “**” operator



```
In [3]: 8 / 5

Out[3]: 1.6

In [4]: 8//5

Out[4]: 1

In [5]: 17 % 3

Out[5]: 2

In [6]: 5 ** 2

Out[6]: 25
```

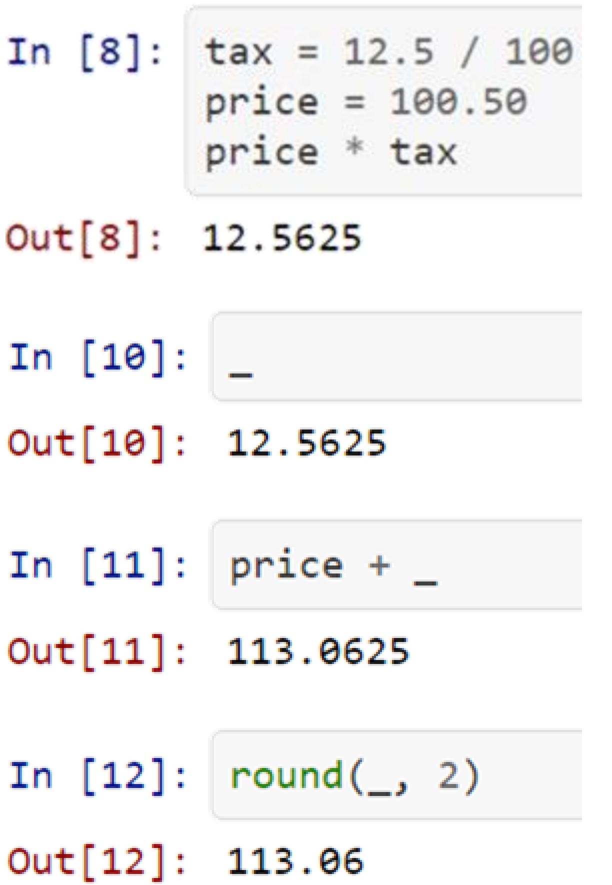
Variables

- The equal sign (=) is used to assign a value to a variable.
- If a variable is not “defined” (assigned a value), trying to use it will give you an error



```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-ab0680a89434> in <module>
----> 1 n

NameError: name 'n' is not defined
```

- You can use python as a desk calculator.
  - ➙In interactive mode, the last printed expression is assigned to the variable _



```
In [8]:  tax = 12.5 / 100
         price = 100.50
         price * tax

Out[8]:  12.5625

In [10]:  _

Out[10]:  12.5625

In [11]:  price + _

Out[11]:  113.0625

In [12]:  round(_, 2)

Out[12]:  113.06
```

Basic Objects and Structures - Numbers

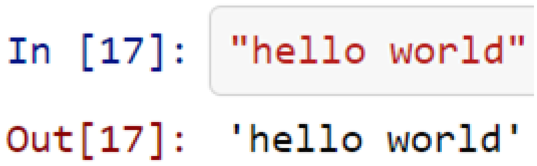
- Integer
  - ➙-3, -2, -1, 0, 1,2,3, ….
- Float
  - ➙Decimal numbers
- Bool
  - ➙Boolean True or False (1 or 0)
- Complex
  - ➙a + bi, a is called the real part, and b is called the imaginary part.
  - ➙abstract quantities that can result in physically meaningful solutions

Basic Objects and Structures – Container Objects

Container	Delimited by
Strings	“”
Lists	[]
Tuples	()
Set	{}
Dictionary	{‘key’: value}

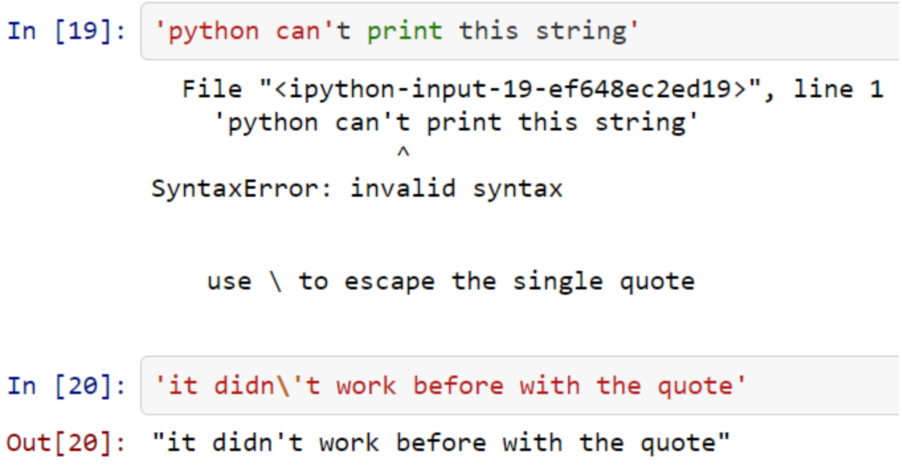
Strings

- Enclosed in single quotes ('...') or double quotes ("...") with the same result



```
In [17]: "hello world"

Out[17]: 'hello world'
```

- use \ to escape the single quote or especial characters “\n”
  - ➞ ‘What’s new in python?’ will generate an error
  - ➞ ‘What\’s new in python?’

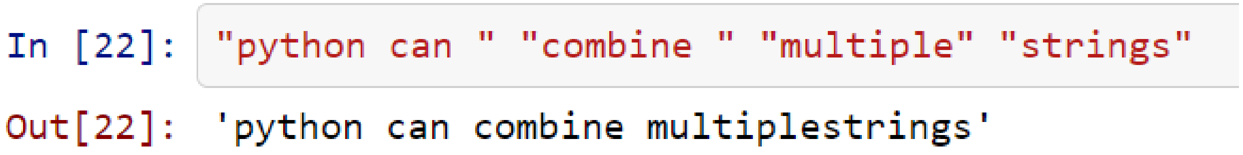


```
In [19]: 'python can't print this string'

  File "<ipython-input-19-ef648ec2ed19>", line 1
    'python can't print this string'
               ^
SyntaxError: invalid syntax
```

use \ to escape the single quote

```
In [20]: 'it didn\'t work before with the quote'

Out[20]: "it didn't work before with the quote"
```

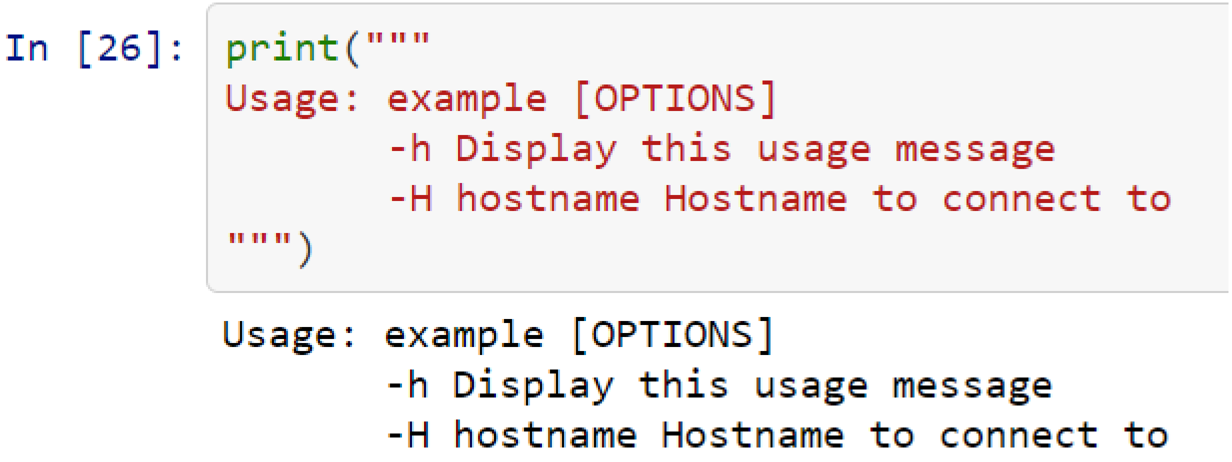
- python can combine multiple strings
  - ➞"python can " "combine " "multiple" "strings"



```
In [22]: "python can " "combine " "multiple" "strings"

Out[22]: 'python can combine multiplestrings'
```

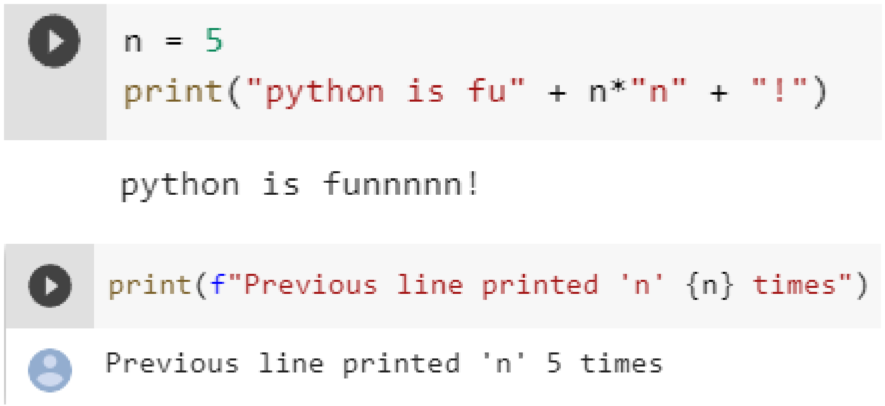
Strings

- String literals can span multiple lines. Another way is using triple-quotes: """..."""
- Integers can be combined with strings to display multiple characters
- String literals prefixed with 'f' or 'F' are commonly called “f-strings” which is short for formatted string literals.
  - ➞ Better Readability
  - ➞ Concise
  - ➞ Less prone to error



```
In [26]: print("""
Usage: example [OPTIONS]
       -h Display this usage message
       -H hostname Hostname to connect to
""")
```

```
Usage: example [OPTIONS]
       -h Display this usage message
       -H hostname Hostname to connect to
```



```
n = 5
print("python is fu" + n*"n" + "!")
```

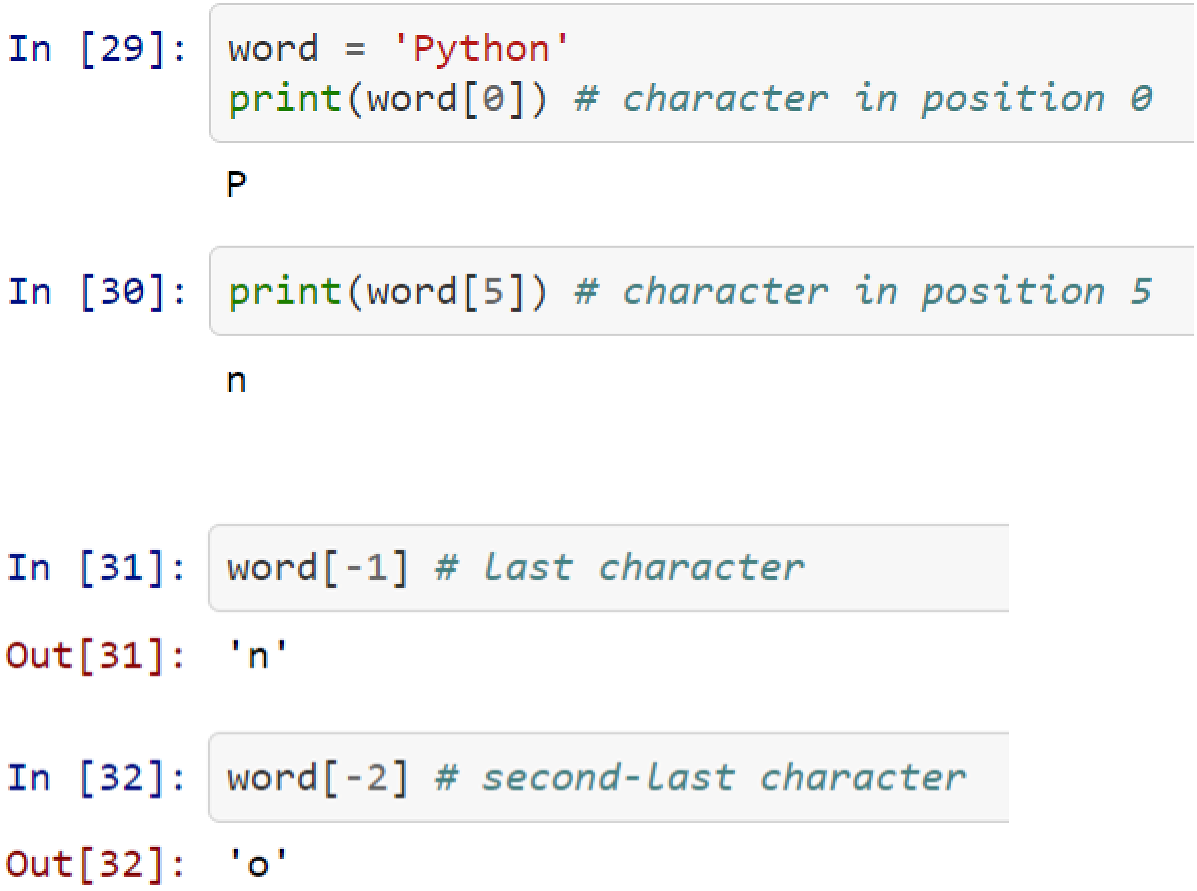
```
python is funnnnn!
```

```
print(f"Previous line printed 'n' {n} times")
```

```
Previous line printed 'n' 5 times
```

Indexing Strings

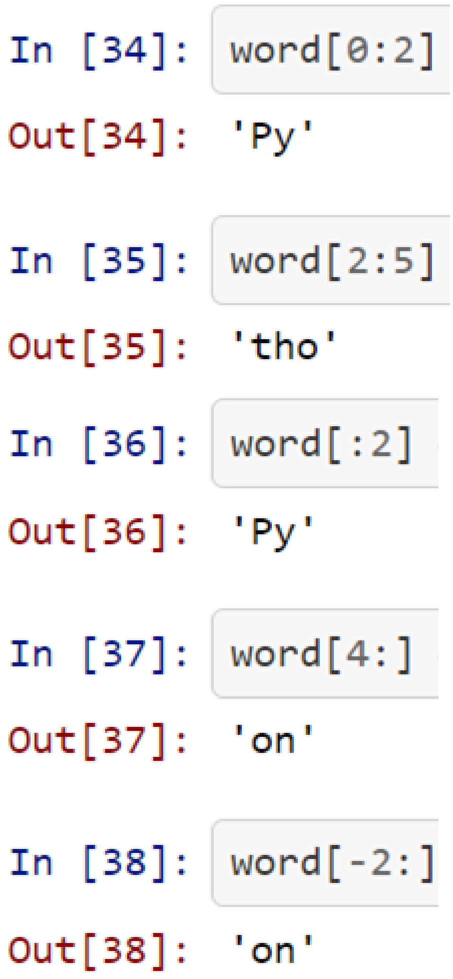
- Strings can be indexed (subscripted), with the first character having index 0
- Indices may also be negative numbers



```
In [29]: word = 'Python'
         print(word[0]) # character in position 0

         P

In [30]: print(word[5]) # character in position 5

         n

In [31]: word[-1] # last character

Out[31]: 'n'

In [32]: word[-2] # second-last character

Out[32]: 'o'
```

Indexing Strings

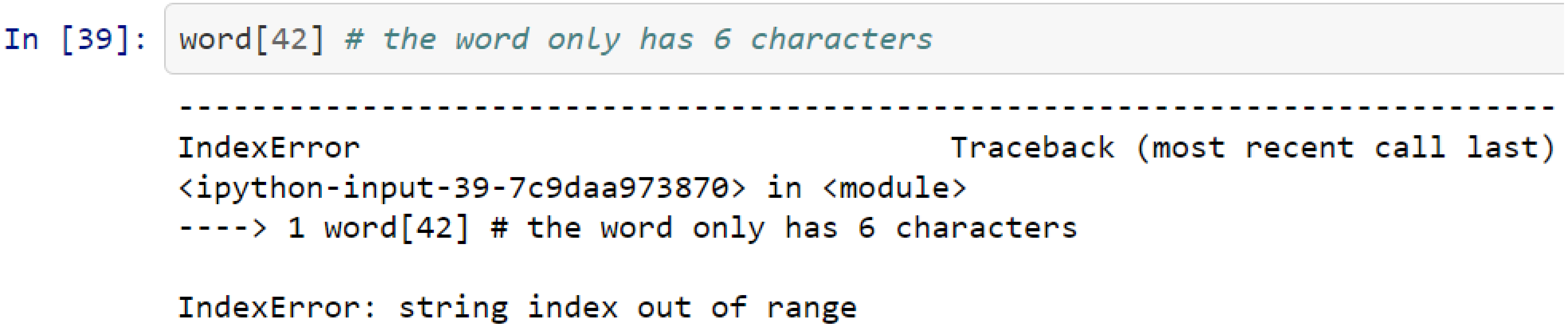
- In addition to indexing, slicing is also supported
  - ➞slicing allows you to obtain substring
- Slice indices have useful defaults
  - ➞an omitted first index defaults to zero,
  - ➞an omitted second index defaults to the size of the string being sliced



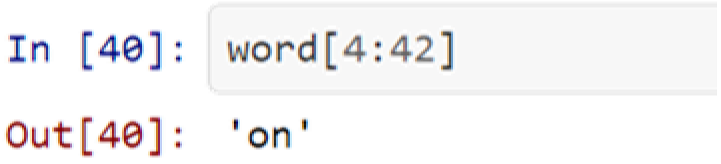
```
In [34]: word[0:2]
Out[34]: 'Py'

In [35]: word[2:5]
Out[35]: 'tho'

In [36]: word[:2]
Out[36]: 'Py'

In [37]: word[4:]
Out[37]: 'on'

In [38]: word[-2:]
Out[38]: 'on'
```

Indexing Errors

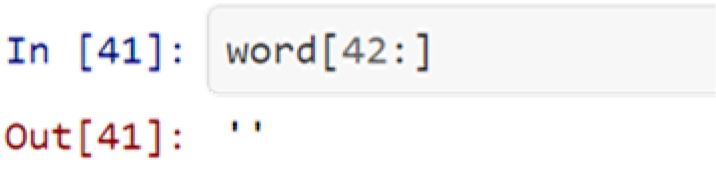
- Attempting to use an index that is too large will result in an error
  - ➞However, out of range slice indexes are handled gracefully when used for slicing



```
In [39]: word[42] # the word only has 6 characters

         ---------------------------------------------------------------------------
         IndexError                                Traceback (most recent call last)
         <ipython-input-39-7c9daa973870> in <module>
         ----> 1 word[42] # the word only has 6 characters

         IndexError: string index out of range
```



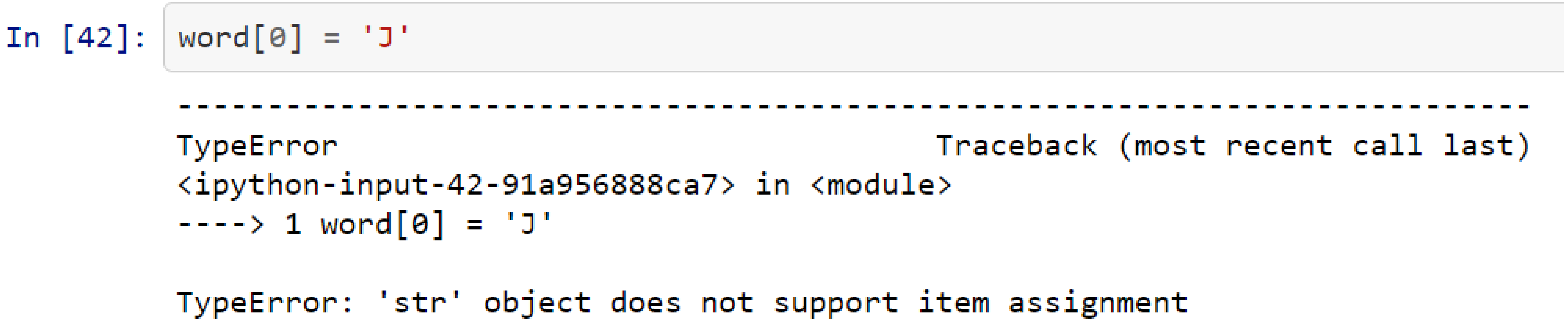
```
In [40]: word[4:42]

Out[40]: 'on'
```



```
In [41]: word[42:]

Out[41]: ''
```

Strings are immutable

- Python strings cannot be changed — they are immutable
  - ➞ Therefore, assigning to an indexed position in the string results in an error

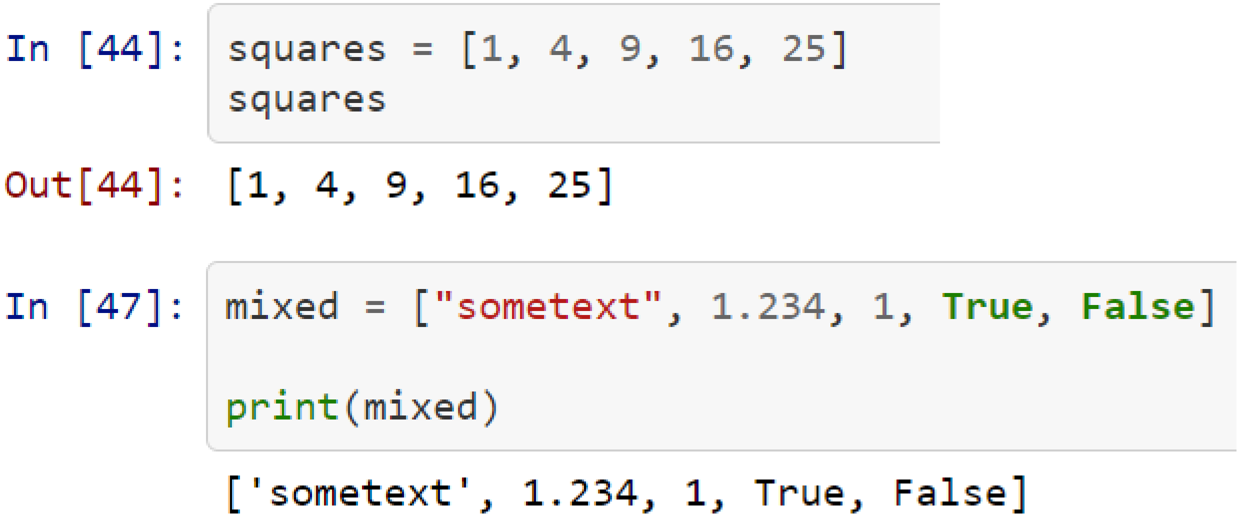


```
In [42]: word[0] = 'J'

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-42-91a956888ca7> in <module>
----> 1 word[0] = 'J'

TypeError: 'str' object does not support item assignment
```

If you need a different string, you should create a new one:

List

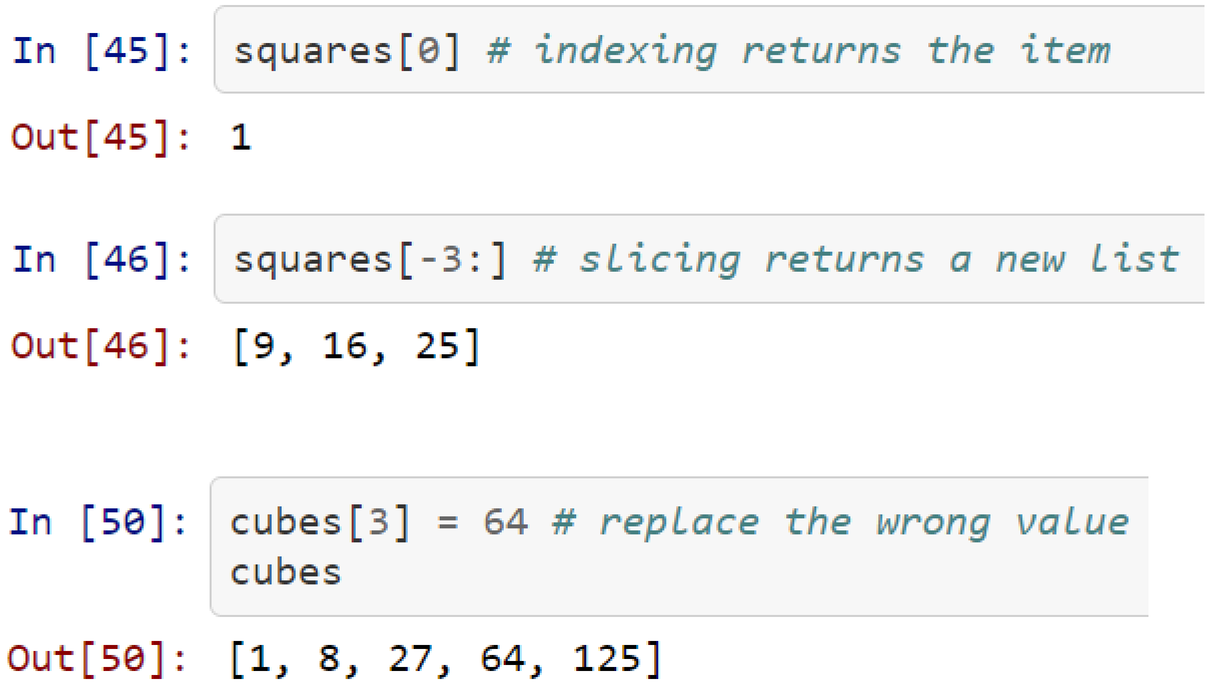
- Lists can be written as a list of comma-separated values (items) between square brackets
- Lists might contain items of different types, but usually the items all have the same type



```
In [44]: squares = [1, 4, 9, 16, 25]
         squares

Out[44]: [1, 4, 9, 16, 25]

In [47]: mixed = ["sometext", 1.234, 1, True, False]

         print(mixed)

         ['sometext', 1.234, 1, True, False]
```

List

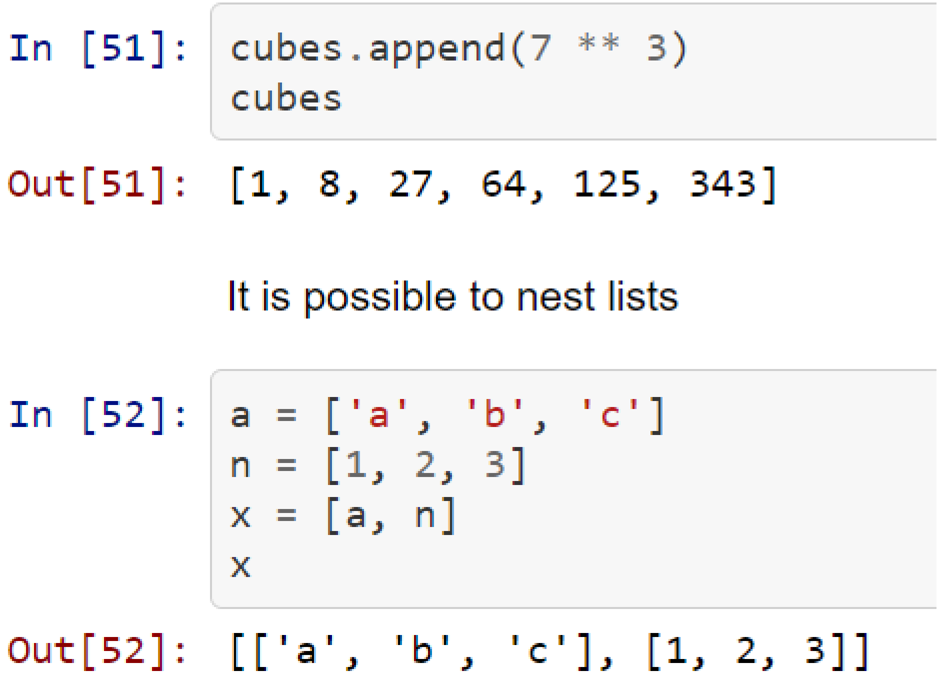
- Like strings (and all other built-in sequence types), lists can be indexed and sliced
- Unlike strings, which are immutable, lists are a mutable type
  - ➞it is possible to change their content



```
In [45]: squares[0] # indexing returns the item

Out[45]: 1

In [46]: squares[-3:] # slicing returns a new list

Out[46]: [9, 16, 25]


In [50]: cubes[3] = 64 # replace the wrong value
         cubes

Out[50]: [1, 8, 27, 64, 125]
```

List

- You can also add new items at the end of the list, by using the append() method
- It is possible to nest lists



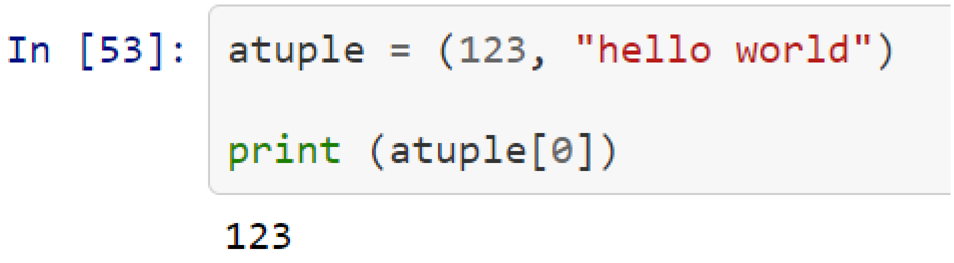
```
In [51]:  cubes.append(7 ** 3)
          cubes

Out[51]:  [1, 8, 27, 64, 125, 343]
```

It is possible to nest lists

```
In [52]:  a = ['a', 'b', 'c']
          n = [1, 2, 3]
          x = [a, n]
          x

Out[52]:  [['a', 'b', 'c'], [1, 2, 3]]
```

Tuples

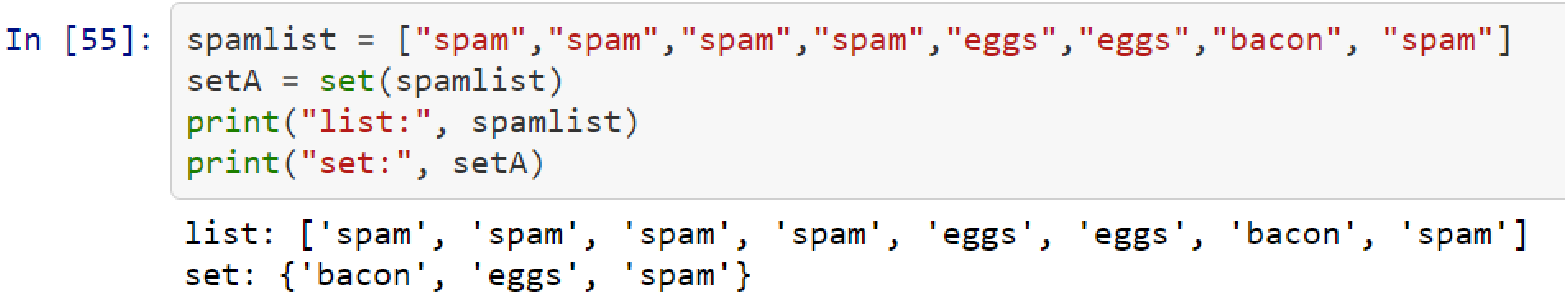
- Tuples Like a list but immutable.
- Makes your code safer in protecting data that does not need to be changed.
- This also means that tuples are very efficient with regards to memory consumption and runtime.
- Tuples are immutable



```
In [53]: atuple = (123, "hello world")

         print (atuple[0])

123
```

Set

- A mathematical set
  - ➞can be used to filter out duplicates in lists
  - ➞check for membership.

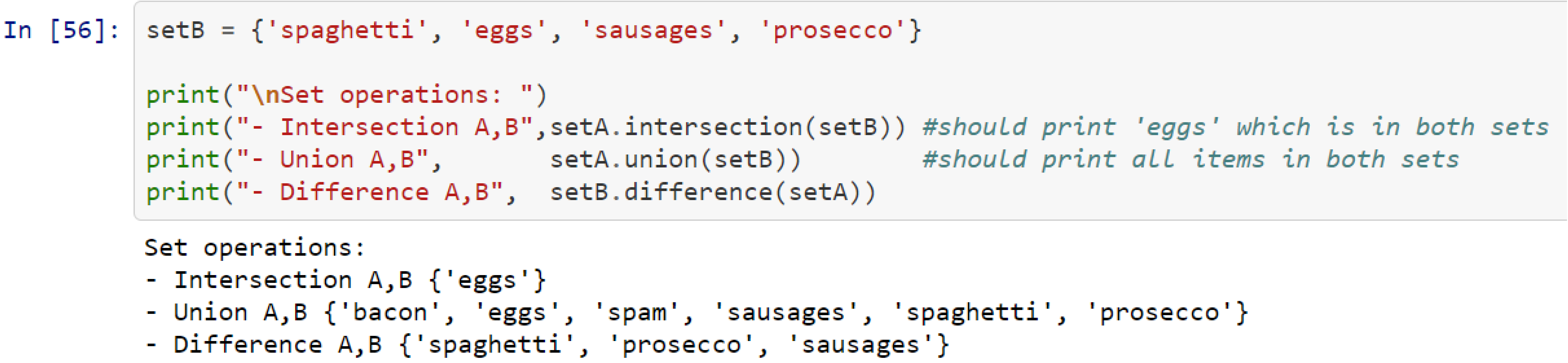


```
In [55]: spamlist = ["spam","spam","spam","spam","eggs","eggs","bacon", "spam"]
         setA = set(spamlist)
         print("list:", spamlist)
         print("set:", setA)
```

```
list: ['spam', 'spam', 'spam', 'spam', 'eggs', 'eggs', 'bacon', 'spam']
set: {'bacon', 'eggs', 'spam'}
```

Set Operations

- Intersection
- Union
- Difference

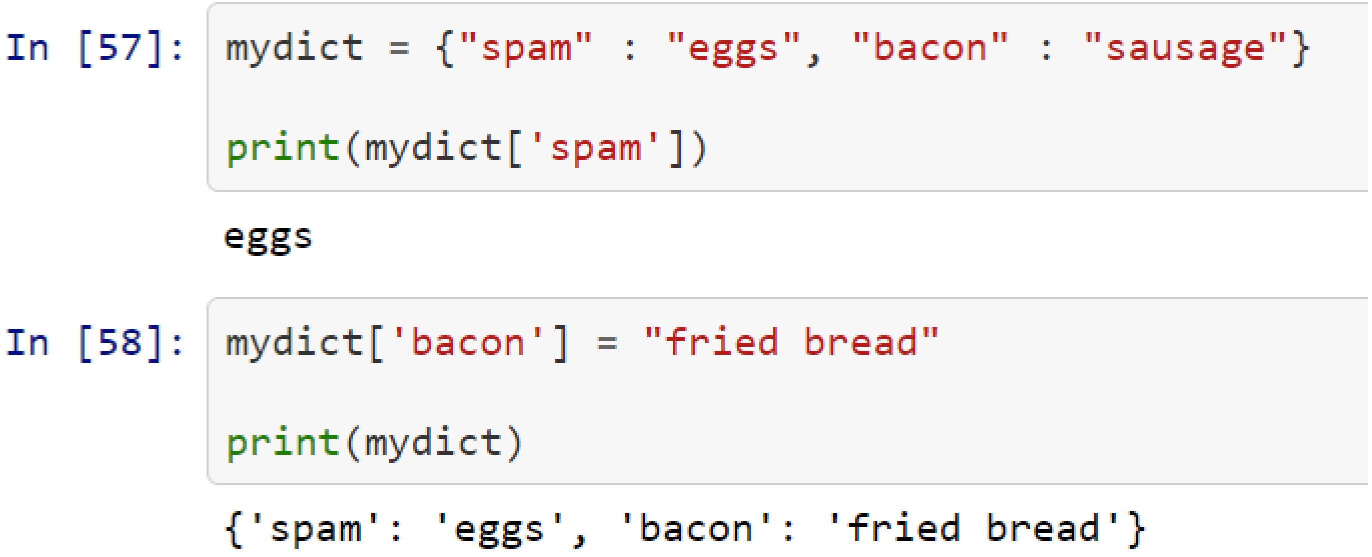


```
In [56]: setB = {'spaghetti', 'eggs', 'sausages', 'prosecco'}

         print("\nSet operations: ")
         print("- Intersection A,B",setA.intersection(setB)) #should print 'eggs' which is in both sets
         print("- Union A,B",       setA.union(setB))        #should print all items in both sets
         print("- Difference A,B",  setB.difference(setA))
```

```
Set operations:
- Intersection A,B {'eggs'}
- Union A,B {'bacon', 'eggs', 'spam', 'sausages', 'spaghetti', 'prosecco'}
- Difference A,B {'spaghetti', 'prosecco', 'sausages'}
```

Dictionary

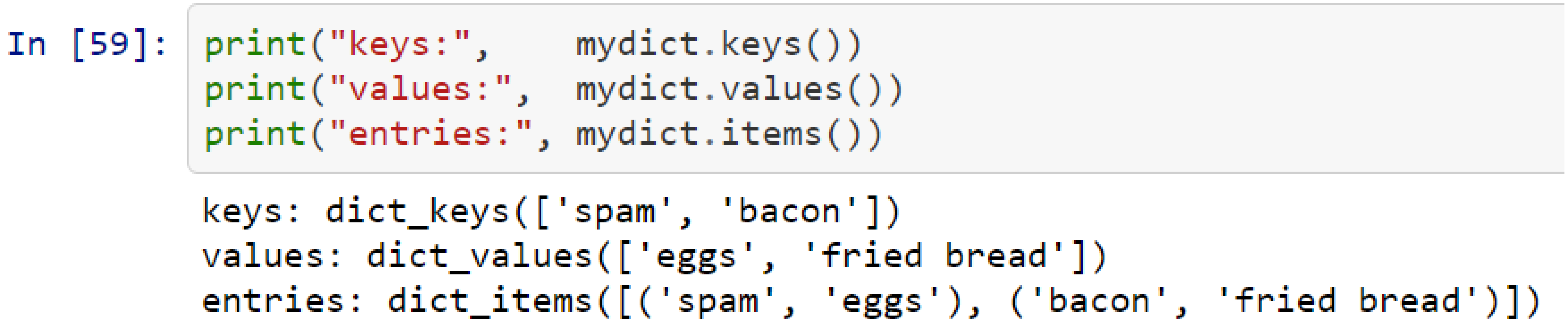
- a mutable set of key -> value pairs maintained in memory
  - ➞{‘key1’: value1, ‘key2’: value2, …}
- allows you to access the items using the indexing syntax and a key



```
In [57]: mydict = {"spam" : "eggs", "bacon" : "sausage"}

         print(mydict['spam'])

eggs

In [58]: mydict['bacon'] = "fried bread"

         print(mydict)

{'spam': 'eggs', 'bacon': 'fried bread'}
```

Dictionary

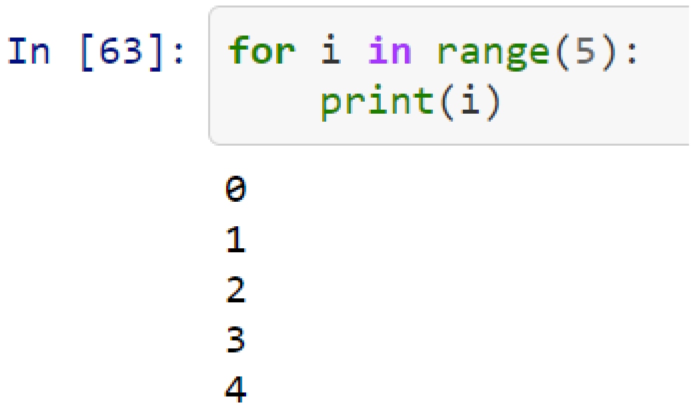
- You can get a list of keys or a list of values
    - even as a tuple



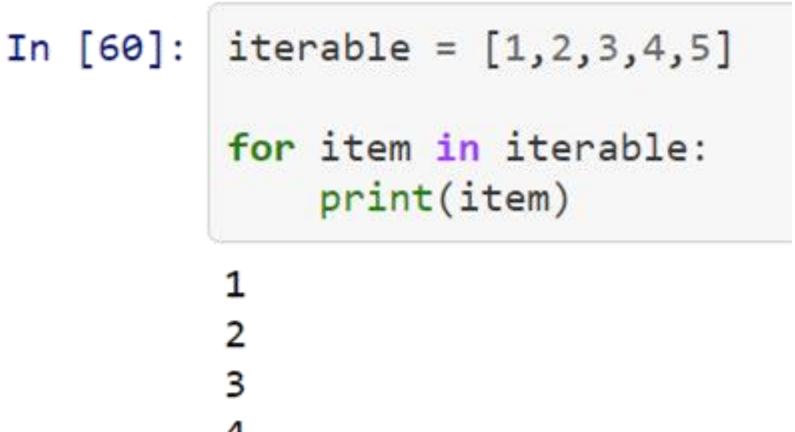
```
In [59]: print("keys:",    mydict.keys())
         print("values:",  mydict.values())
         print("entries:", mydict.items())

         keys: dict_keys(['spam', 'bacon'])
         values: dict_values(['eggs', 'fried bread'])
         entries: dict_items([('spam', 'eggs'), ('bacon', 'fried bread')])
```

Loops and Iterations – For Loop

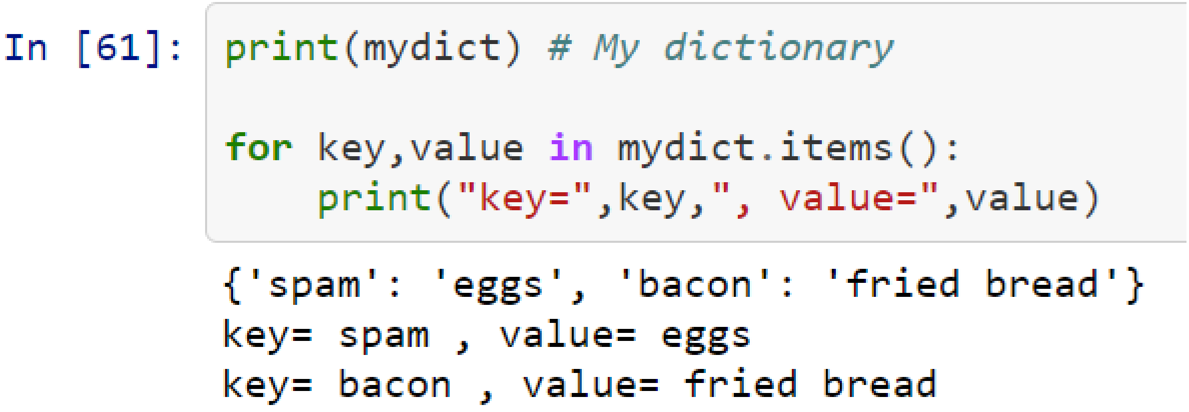
- A basic for loop looks like this



```
In [63]: for i in range(5):
             print(i)

0
1
2
3
4
```



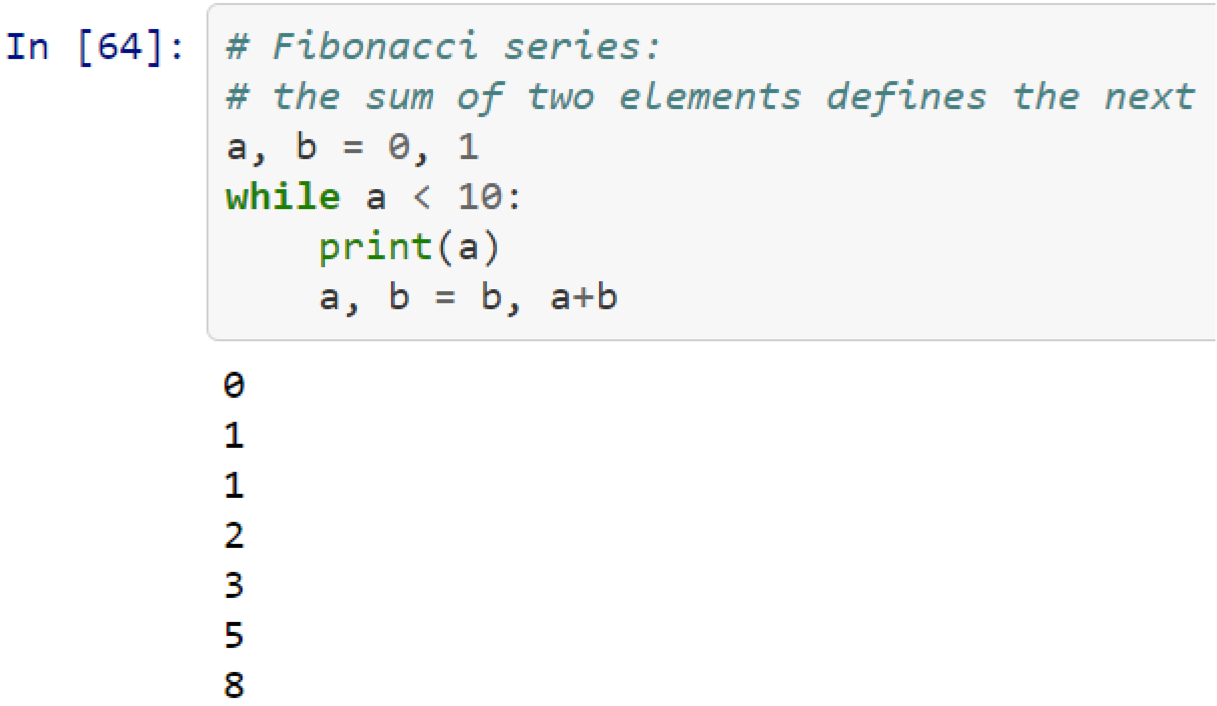
```
In [60]: iterable = [1,2,3,4,5]

         for item in iterable:
             print(item)

1
2
3
4
5
```



```
In [61]: print(mydict) # My dictionary

         for key,value in mydict.items():
             print("key=",key,", value=",value)

{'spam': 'eggs', 'bacon': 'fried bread'}
key= spam , value= eggs
key= bacon , value= fried bread
```

While Loop

- The while loop executes as long as the condition (here: a < 10) remains true.



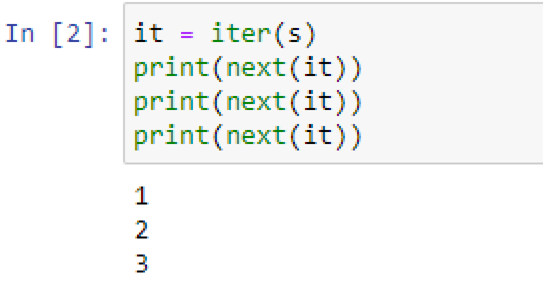
In [64]:

```
# Fibonacci series:
# the sum of two elements defines the next
a, b = 0, 1
while a < 10:
    print(a)
    a, b = b, a+b
```

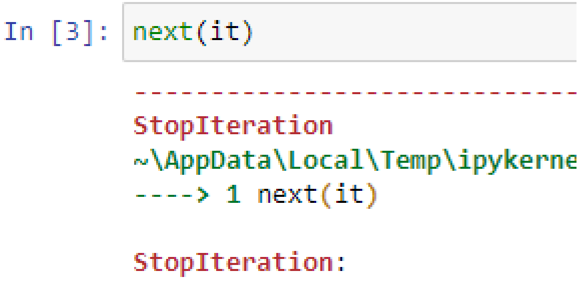
```
0
1
1
2
3
5
8
```

Iterators

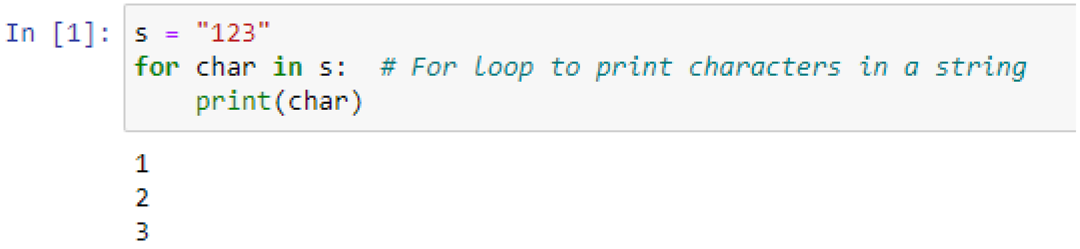
- Behind the scenes, the for statement calls `iter()` on the container object.
- The method `next()` accesses elements in the container one at a time.
- When there are no more elements, `next()` raises a `StopIteration`



```
In [2]: it = iter(s)
        print(next(it))
        print(next(it))
        print(next(it))

        1
        2
        3
```



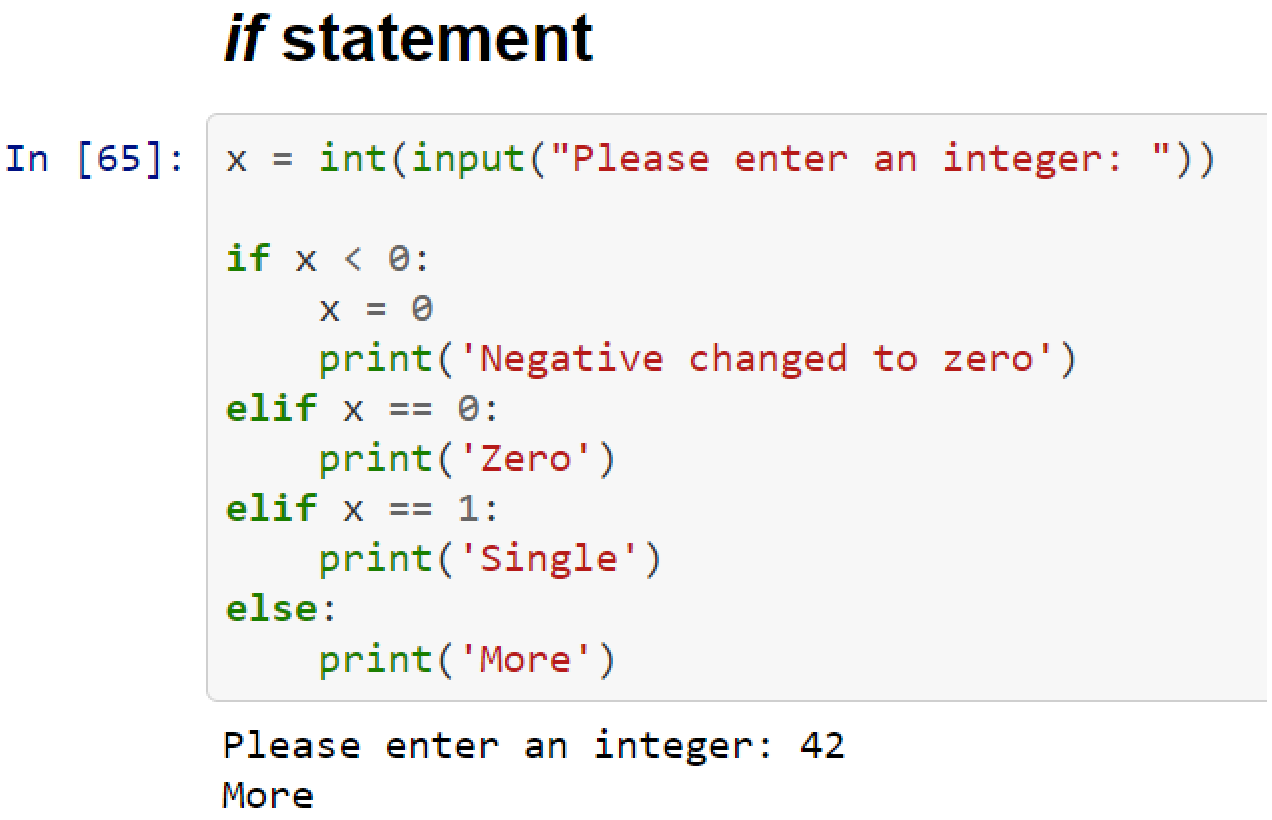
```
In [3]: next(it)

        ------------------------------
        StopIteration
        ~\AppData\Local\Temp\ipykerne
        ----> 1 next(it)

        StopIteration:
```



```
In [1]: s = "123"
        for char in s:  # For loop to print characters in a string
            print(char)

        1
        2
        3
```

if statement

```
if raining
        do not water
else
        do water
```



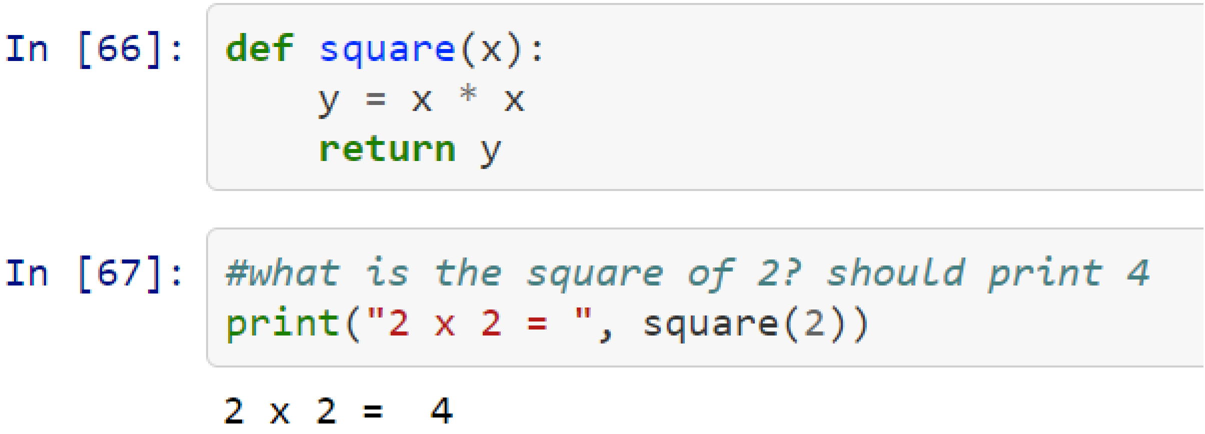
***if* statement**

```
In [65]: x = int(input("Please enter an integer: "))

         if x < 0:
             x = 0
             print('Negative changed to zero')
         elif x == 0:
             print('Zero')
         elif x == 1:
             print('Single')
         else:
             print('More')
```

```
Please enter an integer: 42
More
```

Functions

- Function definitions in Python are very simple.

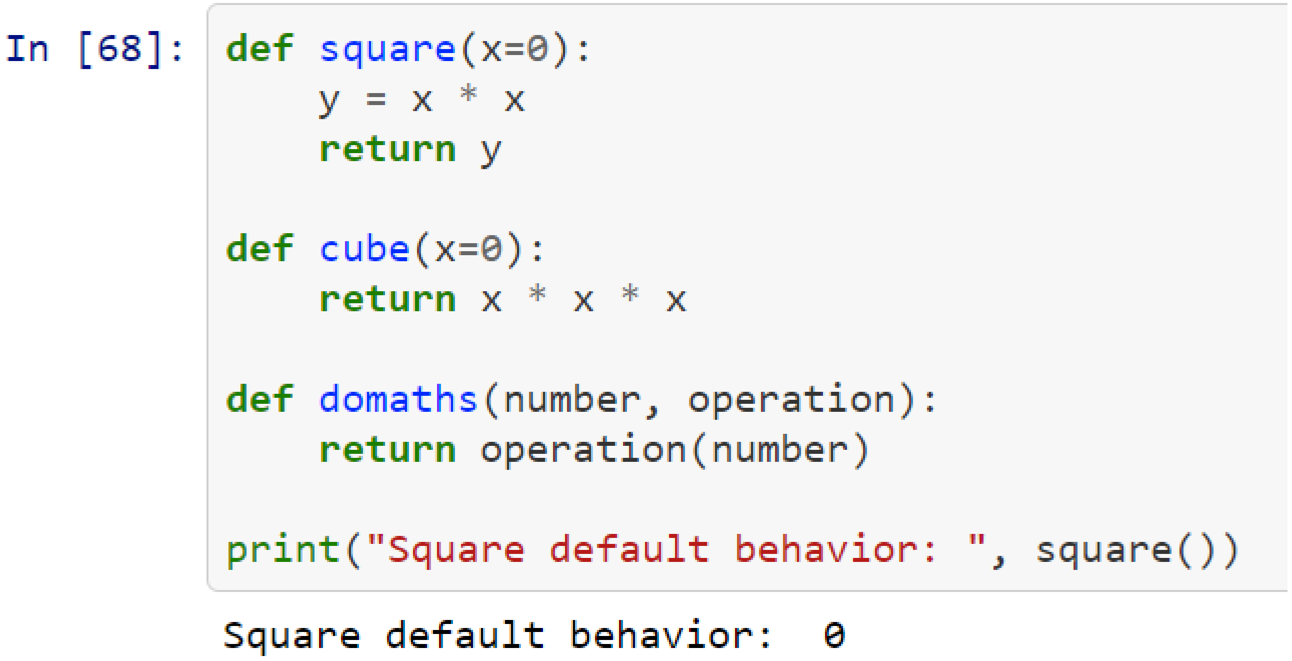


```
In [66]: def square(x):
             y = x * x
             return y
```

```
In [67]: #what is the square of 2? should print 4
         print("2 x 2 = ", square(2))

         2 x 2 =  4
```

Functions

- Functions can have many parameters and default values

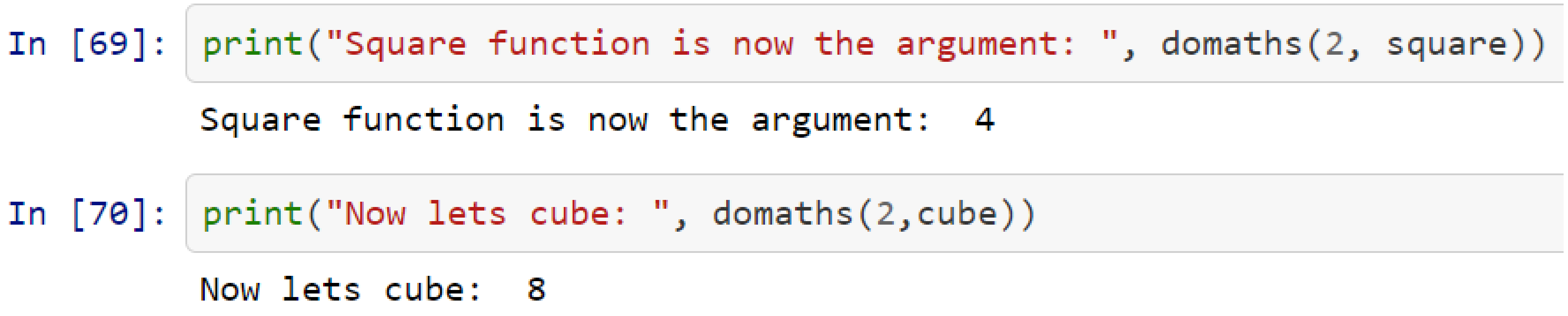


```
In [68]: def square(x=0):
             y = x * x
             return y

         def cube(x=0):
             return x * x * x

         def domaths(number, operation):
             return operation(number)

         print("Square default behavior: ", square())
```

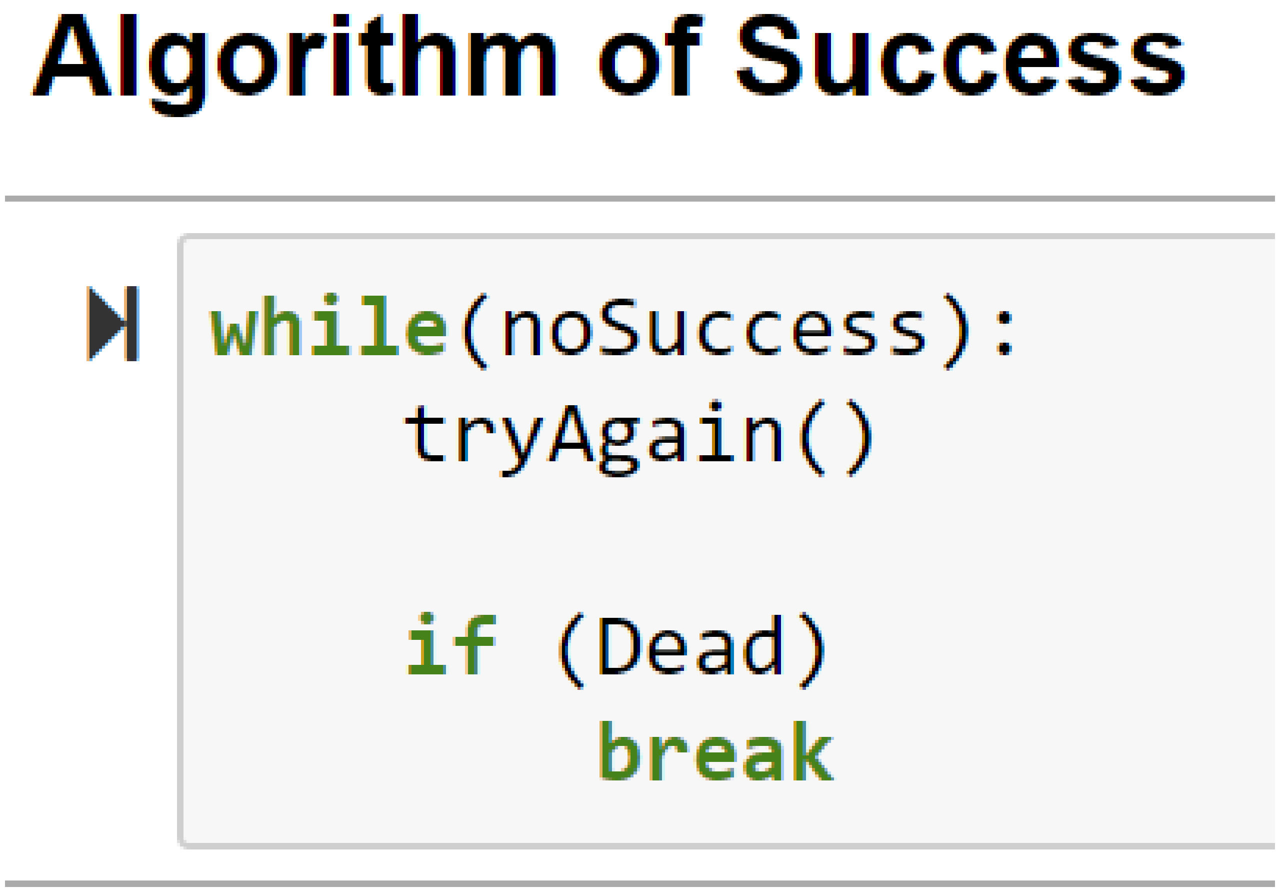
```
Square default behavior:  0
```

Functions

- Functions can have many parameters and default values



```
In [69]: print("Square function is now the argument: ", domaths(2, square))

         Square function is now the argument:  4

In [70]: print("Now lets cube: ", domaths(2,cube))

         Now lets cube:  8
```



Algorithm of Success

```
while(noSuccess):
    tryAgain()

    if (Dead)
        break
```